ARTICLE IN PRESS

Available at www.ComputerScienceWeb.com POWERED BY SCIENCE @ DIRECT®

ELSEVIER

Future Generation Computer Systems 1010 (2003) 1–15

FGCS FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

Grid-enabled particle physics event analysis: experiences using a 10 Gb, high-latency network for a high-energy physics application

W. Allcock [a,*], J. Bresnahan [a], J. Bunn [b], S. Hedge [d], J. Insley [a], R. Kettimuthu [e], H. Newman [b], S. Ravot [b], T. Rimovsky [c], C. Steenberg [b], L. Winkler [a]

[a] *Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*
[b] *California Institute of Technology, Pasadena, CA, USA*
[c] *National Center for Supercomputing Applications, Urbana, IL, USA*
[d] *Illinois Institute of Technology, Chicago, IL, USA*
[e] *The Ohio State University, Columbus, OH, USA*

Abstract

This paper examines issues encountered attempting to exploit a high-bandwidth, high-latency link in support of a high-energy physics (HEP) analysis application. The primary issue was that the TCP additive increase/multiplicative decrease (AIMD) algorithm is not suitable for "long fat networks". While this is a known problem, the magnitude of the impact on application performance was much greater than anticipated. We were able to overcome much of the impact, by altering the AIMD coefficients. Such an approach, of course, is non-TCP compliant, and there was insufficient time to test the network friendliness of these modifications.
© 2003 Published by Elsevier Science B.V.

Keywords: Networks; DataGrid; Congestion avoidance; 10 GigE; Web100

1. Introduction

The major high-energy physics (HEP) experiments of the next 20 years will break new ground in our understanding of the fundamental interactions, structures, and symmetries that govern the nature of matter and space–time. Among the principal goals are to find the mechanism responsible for mass in the universe, the Higgs particles associated with mass generation, and the fundamental mechanism that led to the predominance of matter over antimatter in the observable cosmos.

The largest collaborations today, such as CMS [10] and ATLAS [4], which are building experiments for CERN's Large Hadron Collider (LHC) program [19], each encompass 2000 physicists from 150 institutions in more than 30 countries. Each of these collaborations include 300–400 physicists in the US, from more than 30 universities as well as the major US HEP laboratories. The current generation of operational experiments at SLAC (BaBar [5]) and Fermilab (D0 [11] and CDF [9]), as well as the experiments at the Relativistic Heavy Ion Collider (RHIC) program at BNL [27], faces similar challenges. BaBar in particular has already accumulated datasets approaching a petabyte (1 PB $= 10^{15}$ bytes).

An impression of the complexity of the LHC data can be gained from Fig. 1, which shows simulated

* Corresponding author.
E-mail addresses: allcock@mcs.anl.gov (W. Allcock), kettimuthu.1@osu.edu (R. Kettimuthu).

0167-739X/03/$ – see front matter © 2003 Published by Elsevier Science B.V.
doi:10.1016/S0167-739X(03)00076-1

UNCORRECTED PROOF

ARTICLE IN PRESS

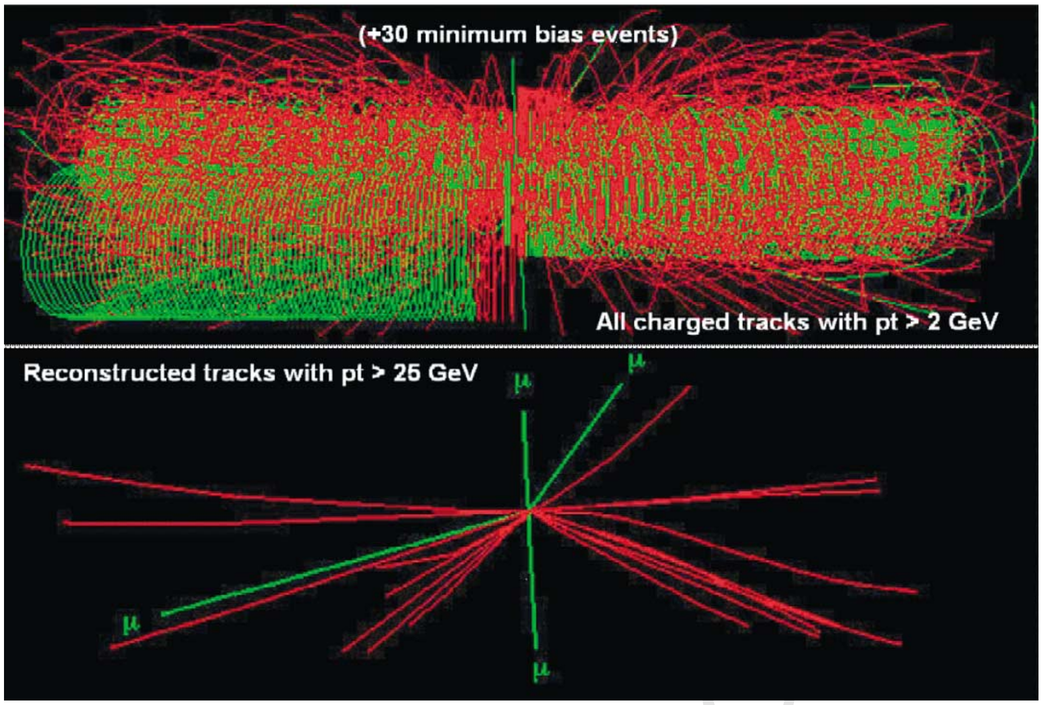
(+30 minimum bias events)
All charged tracks with pt > 2 GeV
Reconstructed tracks with pt > 25 GeV
μ
μ
μ
μ

Fig. 1.

particle trajectories in the inner "tracking" detectors of CMS. The particles are produced in proton–proton collisions that result from the crossing of two proton bunches. A rare proton–proton interaction (approximately 1 in 10^{13}) resulting in the production of a Higgs particle that decays into the distinctive signature of four muons is buried in 30 other "background" interactions produced in the same crossing, as shown in the upper half of the figure. The CMS software filters out the background interactions by isolating the point of origin of the high-momentum tracks in the interaction containing the Higgs particle. This filtering produces the clean configuration shown in the bottom half of the figure. At this point, the mass of the Higgs particle can be deduced from the parameters of its muon decay products, the four straight tracks shown in the figure.

The HEP experiments noted above currently require access to terabyte and petabyte sized datasets and involve massive computations by a geographically distributed set of researchers. These requirements are expected to increase by several orders of magnitude in the next decade. This is representative of a broad class of problems referred to as DataGrid problems [1]. The advent of 10 Gb optical networking is making such analyses practical, at least in theory.

At IGrid2002 [17], a HEP analysis application, Root [28], was used to test the "real-world" feasibility of such analyses. During the experiment, Root accessed large datasets from Chicago over the Starlight Chicago–Amsterdam 10 GigE link using GridFTP as the transport mechanism. A tool called geeViz [2] was used to visualize the transfer on a world map. Unfortunately, the bandwidth achieved during this demonstration was very low, in fact less than that achieved over lower-bandwidth, lower-latency links in the US. Investigation showed that three factors were involved. First, a hardware problem caused wildly varying round trip time (RTT) estimates; such variance reduces the rate at which the TCP sender side congestion window can open. Second, if a packet is refused because the interface queue is full, the Linux TCP implementation considers this to be a congestion event, and the transfer enters congestion avoidance. Third, and much more fundamental, the current additive increase/multiplicative decrease (AIMD) algorithm of the TCP protocol is simply not suitable for bulk data transport on long,

UNCORRECTED PROOF

ARTICLE IN PRESS

fat networks. This issue has been recognized before [12–14,30], but the magnitude of the impact was still surprising. Our results show that standard TCP was unable to achieve better than 200 Mbps per host. Use of the work around daemon (WAD) [34] from the Web100 [35]/Net100 [24] project, however, allowed speeds of 700 Mbps per host by altering the characteristics of the TCP AIMD algorithm. Further testing is needed to determine the "network fair sharing" characteristics with these changes. It should be noted here, that the work around daemon is not available for public distribution due to its potentially serious negative impact on general internet traffic.

2. Overview

The original goal of our IGrid2002 demo was to demonstrate the Root HEP analysis application running over an OC-48 (2.5 Gb) transatlantic network, using the Caltech cluster at StarLight and a portable cluster from Argonne named dusty. Approximately 3 weeks prior to the conference, however, a 10 Gb network link was brought up by Level 3 communications. This provided a second goal, filling this pipe. Unfortunately, it was clear that the hardware we had been planning to use was not sufficient for that task, so we decided to use the 20-node DataGrid cluster at Argonne as a source for an attempt to fully utilize the 10 Gb link. As already noted, performance was very poor, and the majority of our subsequent efforts, revolve around trying to explain this poor performance. Because of hardware issues with dusty, and the heavy usage of the DataGrid cluster, most of the troubleshooting was carried out on the "wonderland" clusters from NCSA, tweedledum at Argonne and madhatter in Amsterdam.

This paper provides details of these efforts. Section 3 describes the hardware and networking, while Section 4 describes the software. Section 5 describes the results of the experiment and our investigations into the underlying reasons for the poor results. Section 6 presents our conclusions and briefly outlines future work.

3. The fabric

In this section we describe the hardware and networking infrastructure used for the experiments. As noted in the overview, multiple clusters from multiple institutions were involved. The various configurations are described below.

3.1. The network

The experimental network constructed for the iGRID2002 event, shown in Fig. 2, enabled host connections at 1 Gbps to an overprovisioned 10 Gbps transatlantic link between Amsterdam and Chicago (6632 km). At Argonne, hosts connected to a 10 Gb switch on loan from Force 10 Networks, which connected to a Juniper T640 exit router. From Argonne, the I-Wire [23] network was used to connect to StarLight [29] in downtown Chicago. At StarLight, another Juniper T640 provided access to a 10 Gb link provided by Level 3 communications to cross the Atlantic and connect to a Cisco 12404 in the Amsterdam POP. There were three additional short 10 Gb hops within the Amsterdam POP to the final hop, a Catalyst 6509 loaned by Cisco Systems, to which hosts in Amsterdam were connected. We used a maximum transmission unit (MTU) value of 1500 bytes. We did not explore larger MTUs because of hardware limitations on the Force 10 switch.

3.2. The DataGrid cluster

One of the sources for the transfer was the DataGrid cluster located at Argonne National Laboratory in Illinois. The cluster is from VA Linux and consists of 20 nodes, with a head node (mayor). Each node is a dual processor 850 MHz, Pentium III (fullon 2 × 2), with 512 MB RAM, one 9 GB system disk, four 50 GB data disks, and an Acenic GA620 fiber Gigabit Ethernet controller. The data disks for each node are in a RAID 0 configuration using Linux software RAID. One-half of the RAID is partitioned for use by PVFS [8,25], resulting in a 2 terabyte shared file system. The Linux 2.4.19 kernel was the operating system installed.

3.3. The dusty cluster

The Distributed Systems Laboratory at Argonne National Laboratory operates three portable eight-node clusters for wide area network testing and development. They are referred to collectively as the Three Amigos, after the comedy movie, and the individual

UNCORRECTED PROOF

ARTICLE IN PRESS

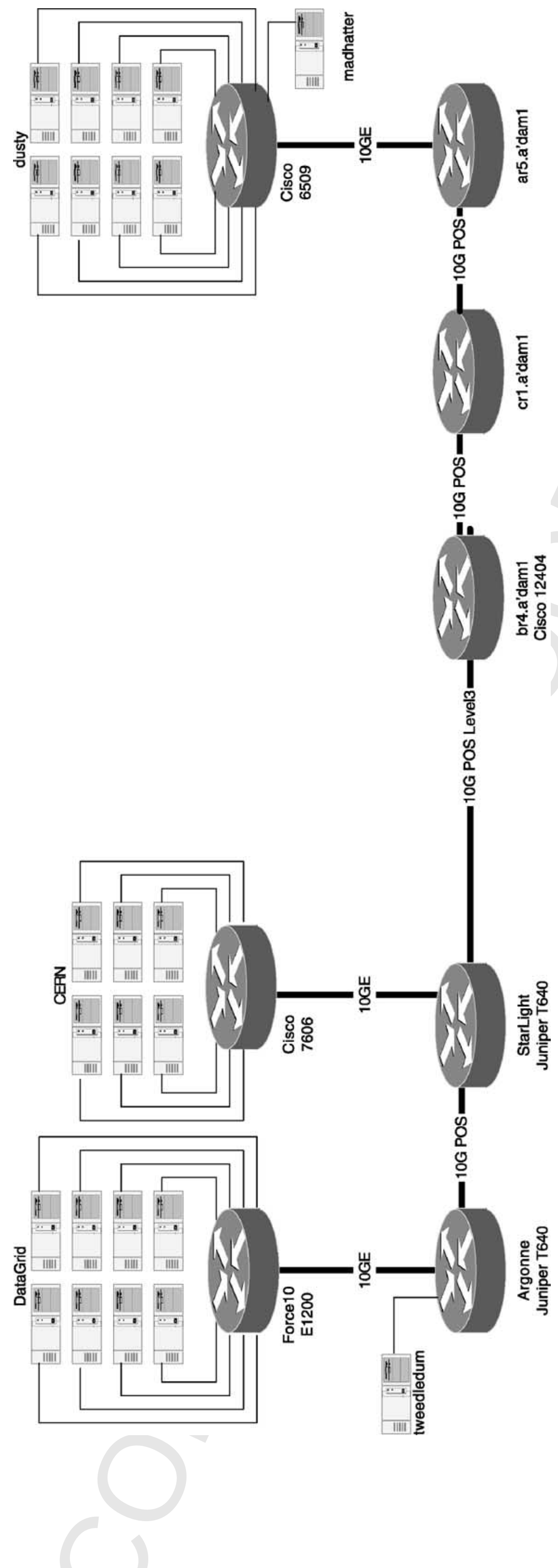

Fig. 2. Network topology.

ARTICLE IN PRESS

clusters bear the names of the lead characters, Lucky, Dusty, and Ned. Dusty was moved to Amsterdam to act as the receiving end of the transfers. Each node in the cluster is a 2U Compaq DL380G2 server with dual 1.13 GHz Pentium IIIs, 2 GB RAM, and six 36 GB SCSI hard drives with onboard HW RAID, using RAID 0, dual redundant power supplies, and redundant cooling fans. Six of the eight nodes in each cluster have a SysKonnect 9821 copper Gigabit NIC. The other two (one of which is the "mayor" or head node) have SysKonnect 9822 dual port copper Gigabit NICs. During the demonstration we installed an additional SysKonnect 9841 fiber NIC. Originally, this was to give us additional potential bandwidth, however, as a result of the problems encountered, we did not use the copper cards for our experiments.

3.4. The Caltech cluster

The six Caltech network systems at StarLight are built from Supermicro P4DP8-G2 motherboards that use 2.2 GHz Intel P4 Xeon processors and come equipped with 1 GB of main memory. The systems are also equipped with SysKonnect SK9843 Gigabit cards in addition to the built-in Intel e1000 Gigabit copper ports.

3.5. The wonderland clusters

The Network Research group at NCSA has a small test cluster for network-based cluster tests. These nodes were distributed at three sites for testing. Eleven are placed at Argonne National Laboratory (tweedledum), eleven remain at the National Center for Supercomputing Applications (tweedledee), and four are installed in Amsterdam (madhatter). The cluster is from Rackable Systems and is comprised of nodes configured with single processor 1 GHz Pentium III CPUs, 256 MB RAM, 40 GB ATA hard drive (ST340016A), and SysKonnect SK-9843 SX Gigabit Ethernet controller.

4. Software

Our planned demo was comprised of three primary software components. The driving application was the HEP analysis program called Root. GridFTP handled data transfer over the network, and visualization of the data transfer was provided by a package called geeViz. These packages are described in more detail below.

4.1. Root

To demonstrate efficient analysis of distributed sets of intermediate-stage data produced by a production effort of CMS institutions, we set up the analysis infrastructure with two distinct elements: downloading data using globus-url-copy, and analysis of downloaded data using the Root analysis package.

The planned demonstration was for the transfer of several data files of the order of 1.7 GB from the Caltech/DOE cluster at the StartLight network POP in Chicago to the cluster at Amsterdam. The data files consisted of 180,000 simulated detector events (or collisions) for the CMS detector, produced by institutions in preparation for the arrival of real detector events when operation of the Large Hadron Collider (LHC) at CERN in Geneva commences. The demonstration machine on the show floor, Rembrandt, mounted a network filesystem volume from the local cluster using PVFS. Data was requested in tandem by the cluster nodes and placed in the network file system volume. The analysis application monitored the network file system for newly transferred files roughly every 2 s while proceeding with the analysis of already received data. In other words, the analysis and data transfer were decoupled, with the former expected to proceed at a much slower rate.

The C++ analysis script maintained a list of downloaded files, over which it iterated, producing a single aggregated result, part of which was displayed on the show floor.

The analysis display consisted of four panels showing three physics quantities: histograms of the number of particles in a so-called particle jets with cone sizes of 0.5 and 0.7 sr, the number of particles in detector towers for a cone size of 0.5, and a scatter plot showing the correlation between the first two quantities. The display was updated for every 50 events analyzed, translating to intervals of roughly 0.5 s. When both the data transfer and analysis was done, the analysis was repeated for the available data files to provide a continuous display on the show floor.

Because the network performance proved to be less than expected, the data transfer portion was scaled

UNCORRECTED PROOF

ARTICLE IN PRESS

down to include only one node at StarLight and Rembrandt, the demonstration machine, was allowed to download the files directly onto the local disk. The analysis application was not altered. Despite the lower data transfer rates, the analysis still proceeded at a slower rate.

This result indicates that more CPUs are needed, even for the simplistic analysis done in this case. Specifically, better resource utilization planning is needed in order to minimize the sum total of the analysis and data transfer times. The file size of the transferred data is in general at least two to three orders of magnitude less than the original datasets, virtually eliminating the constraint of transferring results to their final destination for archiving or further processing.

4.2. GridFTP

GridFTP [16], part of the Globus Toolkit^{TM} [15], is a data transport protocol that provides secure, efficient, and robust data movement in Grid environments. The GridFTP protocol extends the standard FTP protocol and provides a superset of the features offered by the various Grid storage systems currently in use. The GridFTP protocol includes the following:

- Public-key-based Grid Security Infrastructure (GSI) and Kerberos support (both accessible via GSS-API).
- Third-party control of data transfer.
- Parallel data transfer (one host to one host, using multiple TCP streams).
- Striped data transfer (*m* hosts to *n* hosts, possibly using multiple TCP streams if also parallel).
- Manual setting of the TCP buffer size.
- Partial file transfer.
- Support for reliable and restartable data transfer.
- Integrated instrumentation, for monitoring ongoing transfer performance.

Typical transfers on a single node are generally limited to less than 1 Gb because of NIC limitations on commodity hardware today. To overcome this limitation, we deployed a striped server, that is, a data transport mechanism that transferred files many-host to many-host. As noted in the fabric description, one of the transfers was from the 20-node DataGrid cluster at Argonne to a portable 8-node test cluster in Amsterdam. The file was block distributed across all nodes by PVFS. Each node in the striped server read the data local to its host and transmitted that across the network. The receiving nodes then wrote the data to PVFS, and it was again block distributed. The data did not have to be local, since PVFS will do intra-cluster transfers as necessary; but for optimization purposes, we attempted to keep reads and writes local. This technique allows parallelism of CPUs, NICs, disks, and possibly even networks. In our case, on the source side, we had 40 processors, 20 NICs and 20 RAID groups working in parallel. While we did not perform scalability testing, the performance up to 20 nodes was reasonable. The striped server is still a prototype and detailed scalability testing is planned before release.

Normal operation for the FTP protocol closes the data connections after each transfer is complete. This is sub-optimal when multiple transfers between the same two locations are taking place, particularly if they are small files. GridFTP offers the option of “data channel caching”, which holds the connection open for use by subsequent transfers. For this application, a custom service was written that took requests for file transfers and kept the data channel connections open until the next transfer arrived. If it was the same source, destination, and security credentials, the cached channels were utilized.

4.3. geeViz

The GridFTP servers provide performance markers that can be enabled by using a client-side plug-in. During our data transfers we used an additional plug-in to log these performance markers to a specialized logging daemon. To visualize these data transfers, we used geeViz, a tool that integrates application-specific performance data with network routing and connectivity data to create interactive, quasi-real-time visualizations of Grid applications that communicate among other data, the geographical location of application components, the resources that they use, and the nature of the interactions among these components and resources. An interface to the logging daemon allows geeViz to be notified when events of interest are being logged. geeViz can then subscribe to particular streams of events, like the performance markers associated with our GridFTP transfers.

UNCORRECTED PROOF

ARTICLE IN PRESS

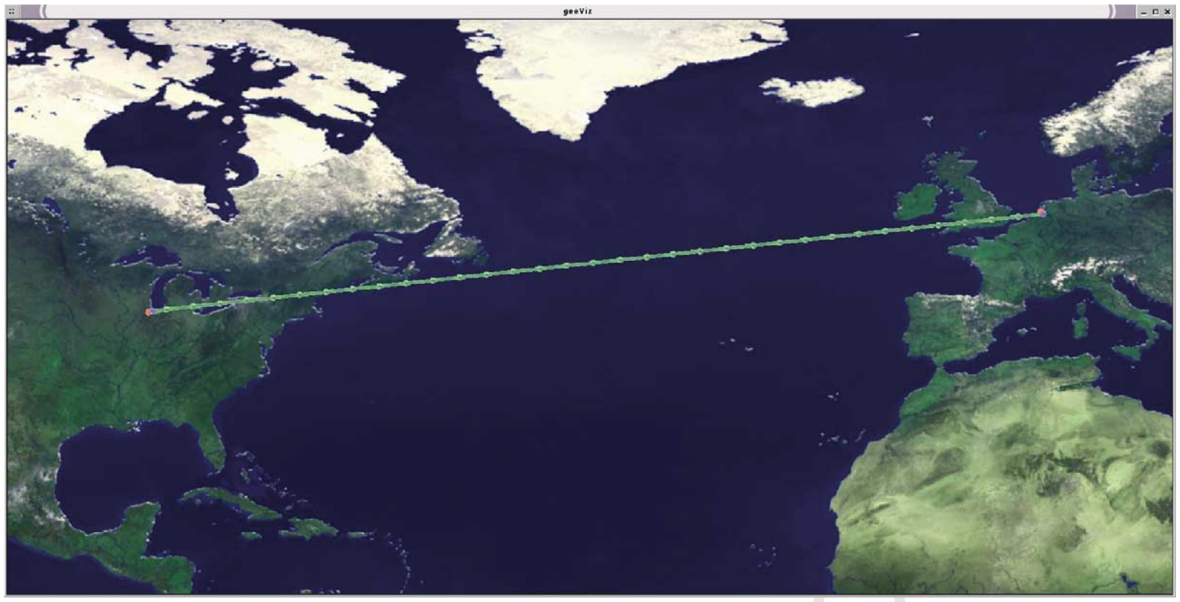

Fig. 3.

When geeViz receives the first transfer event for a stream, it plots the endpoints of the transfer on a map, adds a link between the two sites, and stores the statistics for the transfer (total number of bytes transferred so far, current bandwidth, and average bandwidth). For subsequent events for that transfer, it locates the appropriate sites and link and updates their values. In addition to showing the direct link between source and destination, geeViz allows the user to run a traceroute between the two sites, in order to show the path that data actually takes. Fig. 3 shows our GridFTP transfers from the DataGrid cluster at Argonne to the dusty cluster in Amsterdam. The spheres on the links each represent *N* MB of data and move along the links in the direction of the data flow at a speed relative to the latency on the link, as reported by the traceroute. New spheres are added to the links based on the current bandwidth.

5. Initial results and troubleshooting

In this section we present the initial results obtained during our striped GridFTP transfer. Because these results were significantly lower than expected, we began trying to determine the root cause of this poor performance. This work occurred in several stages. First, we did basic "sanity checks" to ensure this was not a configuration or software problem and to localize which components required further scrutiny. We analyzed tcpdump [31] output using tcptrace [32]. From this analysis, we discovered that we were actually facing three interrelated problems: hardware problems on dusty, an unexpected congestion event resulting from the Linux TCP implementation, and the much more fundamental problem of the TCP additive increase/multiplicative decrease (AIMD) algorithm. Details of these investigations are provided below.

5.1. Initial striped GridFTP transfer results

As discussed above, a transfer was initiated to move the dataset from Chicago to be analyzed locally in Amsterdam, via a striped GridFTP server. Our previous testing had indicated that we could achieve 400 Mb per node. Thus, with 20 nodes participating, we were projecting 8 Gb performance. Early in our testing on the Amsterdam link, however, we realized that we were not able to achieve anywhere near this level of performance. The results from the striped server were somewhat erratic, but hovered around 1 Gb aggregate bandwidth, with 20 nodes and as many as 64 streams per node.

UNCORRECTED PROOF

ARTICLE IN PRESS

5.2. Problem localization

First, we wanted to determine whether GridFTP was the problem. To this end, we used a network testing tool called iperf [22]. Results from iperf were on par with the GridFTP network transfer rates, indicating that GridFTP was not the problem. At this point, we decided to reduce the problem size and focus on single-node, single-stream iperf results as our metric for problem status.

Next, we began basic checks of the network to verify that the Linux ip-sysctl settings were as expected. TCP buffer sizes were checked and were at the desired 12 MB. Selective acknowledgement (SACK) and window scaling were both enabled. We flushed the routing cache before each transfer to ensure that congestion events from previous transfers were not impacting our results. We also consulted with others at the show and determined that applications running over TCP were all experiencing similar problems but that UDP-based applications were not.

At this point, it appeared the problem appeared to be directly related to TCP, so we began to analyze the output from the TCP monitoring tool, tcpdump. A typical output from the TCPtrace tool is shown in Fig. 4. This figure shows that both sides were advertising advanced window scaling, the receiver was advertising a large window scale (10), SACKs were being sent, and there were no lost packets, no retransmissions, and minimal packet reordering, but the sender could not fill the advertised window. Multiple runs were analyzed to and from a combination of hosts, and this behavior was consistent.

We were left with the question of what was limiting the sender. One possibility was that the machines were more heavily CPU loaded that we thought. However, a check of the CPU load showed it to be less than 1%, indicating that the machines were mostly idle and waiting for the network. At this point, we were convinced that the problem lay in the sender-side TCP operation, and we focused our efforts there.

5.3. Sender-side TCP operation

Our demo called for the cluster named dusty to be the receiver in Amsterdam. During our investigations,

UNCORRECTED PROOF

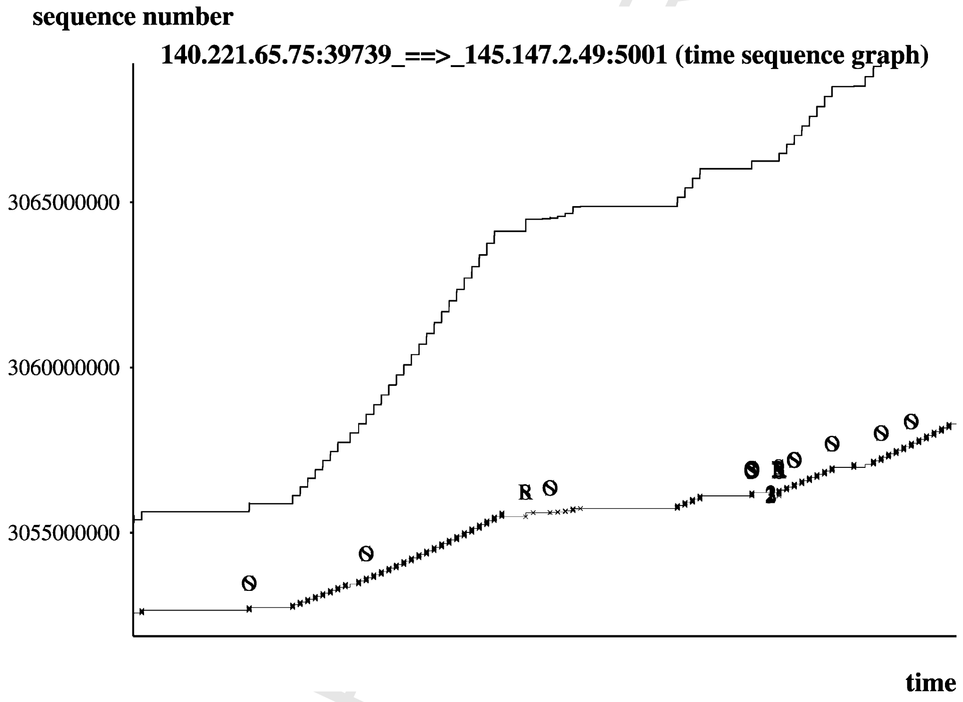

Fig. 4.

ARTICLE IN PRESS

another pair of machines (tweedledum in Chicago and madhatter in Amsterdam), utilizing the same network link were used to determine whether the problem was with the end system or the network path. Almost immediately, we began to see that all machines were both performing badly, but dusty was much worse than madhatter. Since madhatter was getting better and more stable performance, we proceeded with performance improvement testing there, while we tried to understand the operational problems with dusty.

Since the transfer was behaving as though the network was congested, although there were no dropped packets or retransmissions, we tried three approaches to determine the behavior of the system. First, we began a fundamental review of TCP behavior [30]. Second, we installed Web100 kernel patches, as well as the Net100 WAD and tracer daemons to uncover TCP stack performance limitations. Third, we tried to isolate the cause of performance differences between madhatter and dusty.

5.3.1. *Round trip time variance (RTTVAR)*

Using the Web100 tools, we determined that the primary difference between madhatter and dusty was in the round trip time variance (RTTVAR); see Figs. 4 and 5. While RTTs on madhatter hovered consistently around 98 ms, with occasional small variations, on dusty RTTs varied wildly, varying between 98 and >300 ms. This erratic behavior is detrimental because a high RTTVAR will slow the opening of the sender congestion window (CWND) during congestion avoidance and can, though it did not in our case, cause unnecessary retransmission.

Initially, drivers and/or network cards were suspected. By swapping drivers and multiple vendors cards between the machines, however, we were able to isolate the problem to the Compaq servers. They exhibited this behavior with all cards, and those same cards behaved normally in madhatter. So, we removed dusty from the testing and focused on madhatter. We are in contact with Compaq but have not yet discovered the root cause of this problem.

5.3.2. *Linux implementation issue*

While madhatter did not demonstrate the same high RTTVAR with dusty, it did share an apparent anomaly with dusty that was also detrimental to performance. It appeared to be going from slow start into congestion avoidance prematurely. Further investigation revealed that the Linux kernel considers filling the network card interface queue (IFQ) to be a congestion event, and this causes the transfer to enter congestion avoidance. This behavior is referred to as a send-stall. While it is true that network congestion would cause this to happen, in our case that was not the cause. Hence, this behavior was overly aggressive and detrimental to performance.

5.3.3. *TCP additive increase/multiplicative decrease (AIMD) algorithm*

The issues discussed above exacerbated a much more fundamental problem: the additive increase/multiplicative decrease (AIMD) behavior of TCP. Multiple references [12,30] address AIMD behavior in the face of a high-bandwidth, high-latency network, or “long fat networks” (LFNs). Nevertheless, the magnitude of the impact on real-world applications was surprising.

In simplified terms, under congestion, AIMD cuts the current CWND in half and allows it to grow only in a linear fashion equal to one maximum transmission unit (MTU) per round trip time (RTT). While this behavior was adequate for buffer sizes in kilobytes, the large buffers required to fully utilize LFNs means that a single congestion event is devastating. In our case, for 1 Gb with 100 ms latency, the bandwidth-delay product was 12.5 MB.

Floyd [13] states that to fully utilize a 10 Gb link with 100 ms latency, the transfer can only tolerate one dropped packet per 5/3 h. Figs. 6–8 shows a much-simplified graph of how the CWND might grow after a single congestion event. If the congestion event happens shortly after the transfer begins, as in our case due to the Linux IFQ congestion event, it would take on the order of 15 min to achieve peak bandwidth.

Increases in RTT are particularly damaging because RTT impacts two aspects. First, doubling the RTT doubles the value of the bandwidth delay product and thus proportionately double the size of the recovery that must be made in the face of a congestion event. Second, recovery is at the rate of one MTU per RTT; therefore, it takes twice as long to recover. In equation form, it can be presented as follows:

UNCORRECTED PROOF

ARTICLE IN PRESS

```
1 arg remaining, starting with 'oct08_4'
Ostermann's tcptrace -- version 6.2.0 -- Fri Jul 26, 2002

469045 packets seen, 469045 TCP packets traced
elapsed wallclock time: 0:01:20.091167, 5856 pkts/sec analyzed
trace file elapsed time: 0:00:30.854659
TCP connection info:
1 TCP connection traced:
TCP connection 1:
     host a:        140.221.79.2:42159
     host b:        Demo-Jumbo-DHCP-5-128.conf.iGrid2002.org:5001
     complete conn: yes
     first packet:  Tue Oct  8 20:56:39.412883 2002
     last packet:   Tue Oct  8 20:57:10.267542 2002
     elapsed time:  0:00:30.854659
     total packets: 469045
     filename:      oct08_4
a->b:                               b->a:
  total packets:        364680            total packets:        104365
  ack pkts sent:        364679            ack pkts sent:        104365
  pure acks sent:            2            pure acks sent:       104363
  sack pkts sent:            0            sack pkts sent:          410
  max sack blks/ack:         0            max sack blks/ack:         3
  unique bytes sent: 528051840            unique bytes sent:         0
  actual data pkts:     364677            actual data pkts:          0
  actual data bytes: 528051840            actual data bytes:         0
  rexmt data pkts:           0            rexmt data pkts:           0
  rexmt data bytes:          0            rexmt data bytes:          0
  zwnd probe pkts:           0            zwnd probe pkts:           0
  zwnd probe bytes:          0            zwnd probe bytes:          0
  outoforder pkts:         236            outoforder pkts:           0
  pushed data pkts:        452            pushed data pkts:          0
  SYN/FIN pkts sent:       1/1            SYN/FIN pkts sent:       1/1
  req 1323 ws/ts:          Y/Y            req 1323 ws/ts:          Y/Y
  adv wind scale:           10            adv wind scale:           10
  req sack:                  Y            req sack:                  Y
  sacks sent:                0            sacks sent:              410
  urgent data pkts:          0 pkts       urgent data pkts:          0 pkts
  urgent data bytes:         0 bytes      urgent data bytes:         0 bytes
  mss requested:          1460 bytes      mss requested:          1460 bytes
  max segm size:          1448 bytes      max segm size:             0 bytes
  min segm size:           992 bytes      min segm size:             0 bytes
  avg segm size:          1447 bytes      avg segm size:             0 bytes
  max win adv:            5840 bytes      max win adv:         9289728 bytes
  min win adv:            5120 bytes      min win adv:            5120 bytes
  zero win adv:              0 times      zero win adv:              0 times
  avg win adv:            5120 bytes      avg win adv:            7160 bytes
  initial window:         2896 bytes      initial window:            0 bytes
  initial window:            2 pkts       initial window:            0 pkts
  ttl stream length: 601980928 bytes      ttl stream length:         0 bytes
  missed data:        73929088 bytes      missed data:               0 bytes
  truncated data:    515652822 bytes      truncated data:            0 bytes
  truncated packets:    364677 pkts       truncated packets:         0 pkts
  data xmit time:       30.657 secs       data xmit time:        0.000 secs
  idletime max:           98.8 ms         idletime max:           99.2 ms
  throughput:         17114169 Bps        throughput:                0 Bps
```

Fig. 5.

$$\text{recovery time} = \frac{\text{bytes to recover}}{\text{rate of recovery}},$$

$$\text{bytes to recover} \propto \text{RTT} \times \text{BW}$$

$$\Rightarrow \text{bandwidth delay product},$$

$$\text{rate of recovery} \propto \frac{\text{MTU}}{\text{RTT}}$$

Substituting

$$\text{recovery time} \propto \frac{\text{RTT} \times \text{BW}}{\text{MTU}/\text{RTT}} \Rightarrow \frac{\text{RTT}^2 \times \text{BW}}{\text{MTU}}$$

This indicates that doubling the RTT would increase the recovery time by a factor of 4. Also, larger MTUs

UNCORRE

ARTICLE IN PRESS

have a positive effect, but it is linear; doubling the MTU will halve the recovery time.

5.3.4. The work around daemon (WAD)

To further investigate and confirm this behavior, we obtained access to the Net100 WAD. This daemon allows various parameters within the TCP stack to be altered dynamically. The daemon configuration permits specification of tuning parameters per flow (source, source port, destination, destination port). The configuration entries we used for our analysis included the additive increase factor, multiplicative decrease factor and slow start threshold.

Using the WAD allowed us to improve recovery after a loss recovery or congestion event, and hence, throughput, by altering TCP's multiplicative decrease. Normally, TCP reduces CWND by 0.5 after a loss and increases CWND by 1 segment per round trip time. With WAD, we adjusted the multiplicative decrease to 0.0125 and the additive increase to be 4. Our results show significant improvements are achievable. The Net100 kernel extensions allowed us to specify a threshold to switch to Floyd's limited slow-start algorithm [14]. Floyd recommends max_ssthresh to be

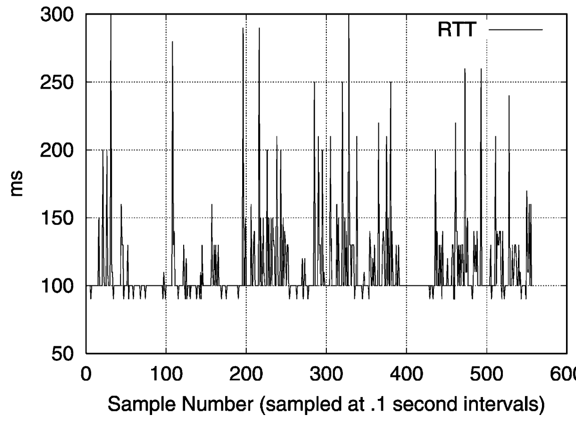

Fig. 6.

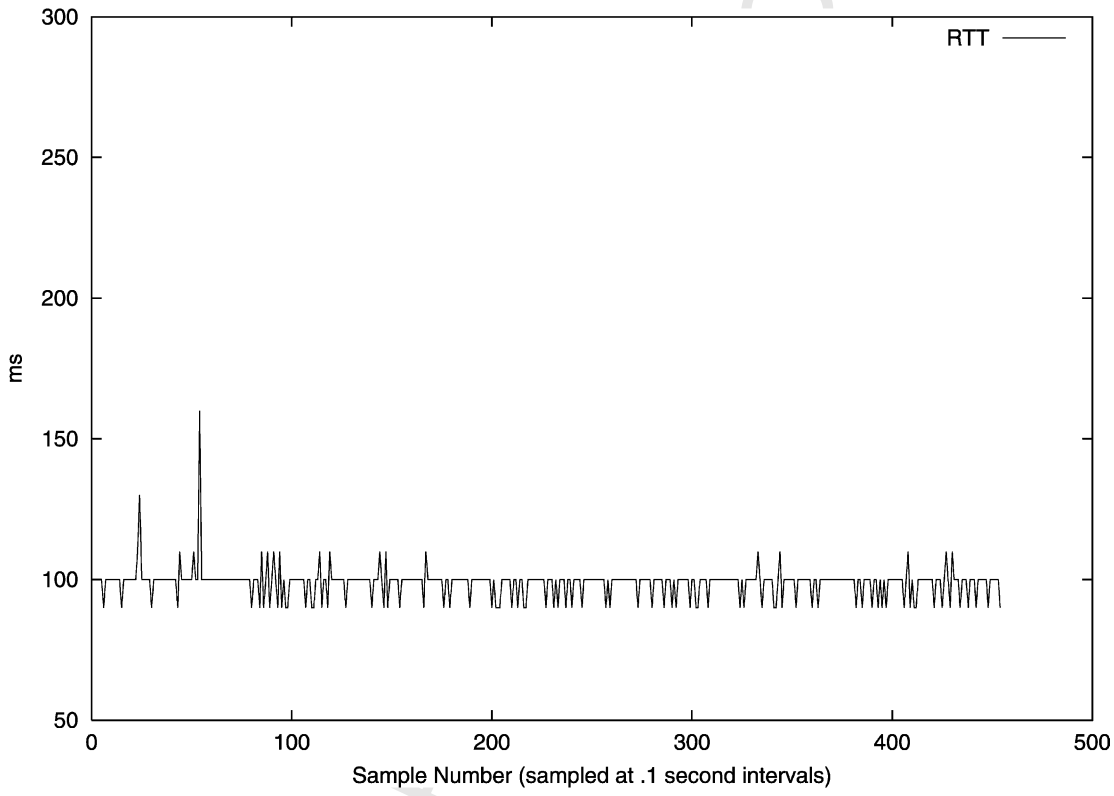

Fig. 7.

UNCORRECTED PROOF

ARTICLE IN PRESS

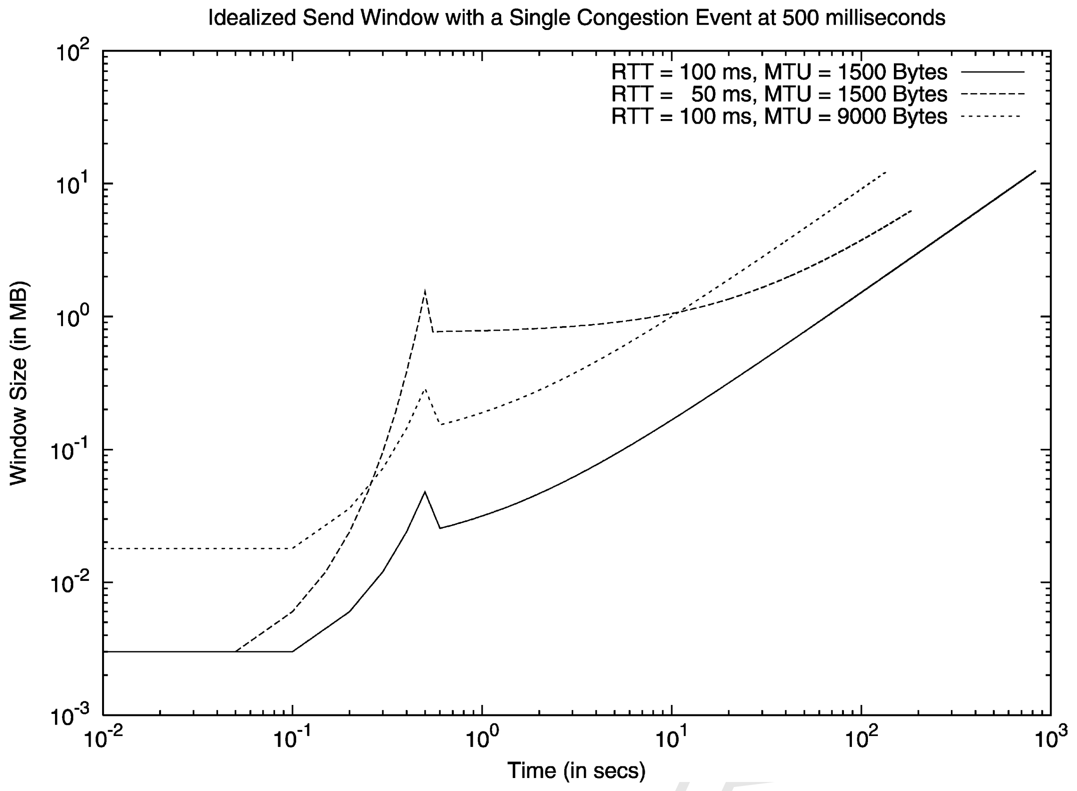

Fig. 8.

set to 100, we chose a more aggressive setting of 200.

Using the WAD, one also has the capability to monitor any Web100 variable for any flow, via the tracer daemon. During these experiments the Net100 tracer daemon was used to poll the kernel every 0.1 s and log Web100 variables to disk. Results showing iperf performance with standard TCP settings for AIMD and with the more aggressive setting listed above, are shown in Figs. 9 and 10.

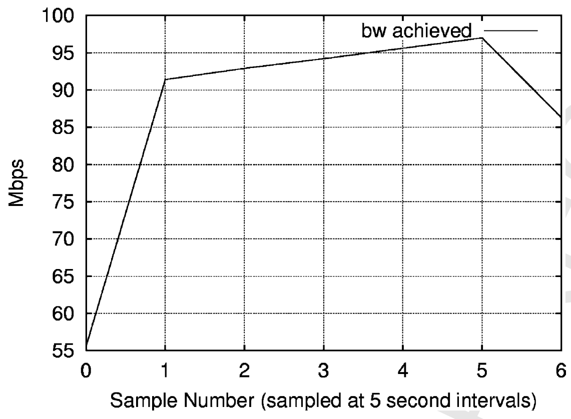

Fig. 9.

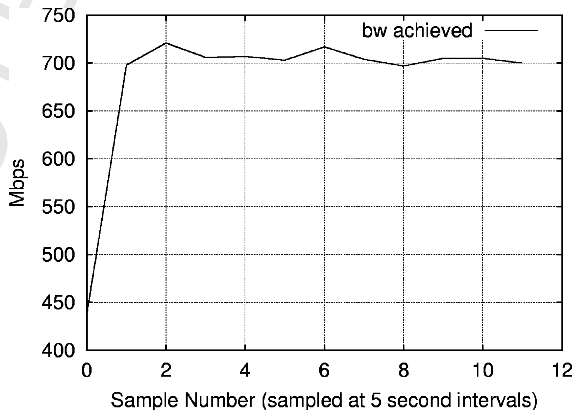

Fig. 10.

UNCORRECTED PROOF

ARTICLE IN PRESS

6. Conclusions and future work

TCP has served the Internet community well for many years. However, it was not designed for bulk data transport at the speeds available today. This fact has been widely discussed in the literature [12,13,30] and multiple attempts have been made to ameliorate this issue [3,7,13,14,18,20,21]. This demonstration was real-world proof of this over a high-bandwidth (10 GigE), high-latency (98 ms) link. We had no network congestion events of any kind, but were still unable to utilize the bandwidth due to the AIMD algorithm of TCP. We were, however, able to show that by making the AIMD algorithm more aggressive higher bandwidth could be achieved. No experiments have yet been run to determine the impact on fair sharing with this more aggressive AIMD, we are unclear the impact it would have on general Internet use. Web100 + WAD provide a transparent mechanisms to expose variables critical to the operation of TCP and for working around network tuning issues. While the development of these tools aims to eliminate the “wizard gap” [36], understanding and effectively using WAD to overcome TCP performance issues still requires an intimate understanding of TCP dynamics.

For this specific experiment, the following items need to be addressed:

- optimization of TCP AIMD parameter impact on network fair sharing;
- effects of jumbo frames in this environment;
- tuning of the interface queue and other contributors to SendStalls.

In the longer term, an alternative to today’s TCP algorithm is needed if we are to gain full advantage from the high-speed optical networks being put in place. Many people are working in this area. They are generally following one of two basic approaches. Sally Floyd and others are trying to use the “evolutionary” approach and make limited changes to the TCP protocol [3,7,13,14,18,20,21]. The other camp is taking the “revolutionary” approach and investigating completely different transport protocols such as R-UDP [6], RBUDP [26], Tsunami [33] and other similar reliable UDP-based transports.

Acknowledgements

Our thanks go out to all the support people who put forth extraordinary efforts to make this 10 GigE link happen, give us access to additional test hardware on the fly, and reconfigured things for us on a moments notice. I would particularly like to thank Charles Bacon for getting on a plane to Amsterdam on 2 h notice when dusty was damaged in shipment.

This work was supported in part by the Mathematical Information and Computational Sciences Division Subprogram of the Office of Advanced Scientific Computing Research, US Department of Energy, under contract W-31-109-Eng-38, also by DOE Grants DE-FG03-92-ER40701 and DE-FC03-99ER25410 and by NSF Grants 8002-48195, PHY-0122557 and ACI-96-19020.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, Data Management and Transfer in High Performance Computational Grid Environments, Parallel Comput. (2001).

[2] W. Allcock, J. Bester, J. Bresnahan, I. Foster, J. Gawor, J.A. Insley, J.M. Link, M.E. Papka, GridMapper: a tool for visualizing the behavior of large-scale distributed systems, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, 16–24 July 2002, pp. 179–187.

[3] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, April 1999 (Request for Comments 2581).

[4] The ATLAS Technical Proposal, CERN/LHCC 94-43 (1994) and CERN LHCC-P2. http://atlasinfo.cern.ch/ATLAS/TP/NEW/HTML/tp9new/tp9.html. Also see the ALICE experiment at http://www.cern.ch/ALICE and the LHCb experiment at http://lhcb-public.web.cern.ch/lhcb-public/.

[5] The BaBar Experiment at SLAC. http://www-public.slac.stanford.edu/babar/.

[6] T. Bova, T. Krivoruchka, Reliable UDP Protocol—Internet Draft, February 1999. draft-ietf-sigtran-reliable-udp-00.txt.

[7] L.S. Brakmo, L.L. Peterson, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE J. Selected Areas Commun. 13 (8) (1995).

[8] P.H. Carns, W.B. Ligon III, R.B. Ross, R. Thakur, PVFS: a parallel file system for Linux clusters, in: Proceedings of the Fourth Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317–327.

[9] The CDF Experiment at Fermilab. http://www-cdf.fnal.gov/.

[10] The Compact Muon Solenoid Technical Proposal, CERN/LHCC 94-38 (1994) and CERN LHCC-P1. http://cmsdoc.cern.ch/.

UNCORRECTED PROOF

ARTICLE IN PRESS

[11] The D0 Experiment at Fermilab. http://www-d0.fnal.gov/.
[12] W. Feng, P. Tinnakornsrisuphap, The failure of TCP in high-performance computational grids, Supercomputing (2000).
[13] S. Floyd, High Speed TCP for Large Congestion Windows, Internet draft draft-floyd-tcp-highspeed-01.txt, work in progress, August 2002.
[14] S. Floyd, Limited Slow-Start for TCP with Large Congestion Windows, Internet draft draft-floyd-tcp-slowstart-01.txt, work in progress, August 2002.
[15] Globus Project. http://www.globus.org/.
[16] Globus Project, GridFTP—Universal Data Transfer for the Grid, White Paper, 5 September 2000.
[17] http://www.igrid2002.org/.
[18] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, May 1992 (Request for Comments 1323).
[19] http://www.cern.ch/LHC.
[20] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, October 1996 (Request for Comments 2018).
[21] M. Mathis, J. Mahdavi, Forward Acknowledgement: Refining TCP Congestion Control, SIGCOMM, August 1996.
[22] Iperf Tool. http://dast.nlanr.net/Projects/Iperf/.
[23] I-Wire Project. http://www.i-wire.org/.
[24] Net100 Project. http://www.net100.org/.
[25] PVFS Project. http://parlweb.parl.clemson.edu/pvfs/.
[26] RBUDP. http://www.evl.uic.edu/paper/pdf/cluster2002.pdf.
[27] The Relativistic Heavy Ion Collider at BNL. http://www.bnl.gov/RHIC/.
[28] The ROOT System. http://root.cern.ch/.
[29] StarLight Project. http://www.startap.net/starlight/.
[30] W.R. Stevens, TCP/IP Illustrated, vol. 1, The Protocols, Addison-Wesley, Reading, MA, 1994.
[31] Tcpdump Tool. http://www.tcpdump.org/.
[32] Tcptrace Tool. http://irg.cs.ohiou.edu/software/tcptrace/tcptrace.html.
[33] Tsunami—a hybrid TCP/UDP based file transfer protocol. http://www.ncne.nlanr.net/training/techs/2002/0728/presentations/pptfiles/200207-wallace1.ppt.
[34] WAD. http://www.csm.ornl.gov/~dunigan/net100/wad.html.
[35] Web100 Project. http://www.web100.org/.
[36] http://ncne.nlanr.net/training/techs/1999/991205/Talks/mathis_991205_Pushing_Up_Performance/sld003.htm.

W. Allcock is the technology coordinator for GridFTP and is a member of the Globus Project. He has a Bachelor of Science degree in Computer Science, with an emphasis in Electronics and Instrumentation from the University of Wisconsin—Oshkosh, and a Master of Science in Paper Science from the Institute of Paper Science and Technology. His research interests center around DataGrid technology and particularly high-speed network transport.

J. Bresnahan received his BS in Computer Science from Northern Illinois University in 1997 and his MS in Computer Science from Northern Illinois University in 1998. He is a senior scientific programmer with the Mathematics and Computer Science Division at Argonne National Laboratory, where he is currently working with the Globus Project as a member of the GridFTP team, designing and implementing data transfer protocols for the Grid.

J. Bunn is a Senior Scientist at the Center for Advanced Computing Research at the California Institute of Technology in Pasadena, USA. He gained his BSc (Hons) in Physics from the University of Manchester in 1980, and his PhD in Experimental Particle Physics from the University of Sheffield in 1983. His research interests include high-performance network and computing systems and Grid architectures that will address the data analysis challenges posed by the Large Hadron Collider experiments at CERN.

S. Hedge is currently doing Masters in Computer Science at Illinois Institute of Technology, Chicago and is a Research Aide at Argonne National Lab. He received Bachelor's degree from Bangalore University, India. His previous work experience was as Software Engineer at The John F. Welch Technology Centre (integral part of GE Global Research) and ANZ Information Technology, for 3 years. Research interests includes distributed computing, fault and performance management in high-performance networks. Currently working with Network Aware GridFTP and Linux Kernel Instrumentation.

J. Insley received his BFA in electronic media from Northern Illinois University in 1991, his MFA in electronic visualization from the University of Illinois at Chicago in 1997, and his MS in computer science from the University of Illinois at Chicago in 2002. He has been a scientific programmer with the Mathematics and Computer Science Division at Argonne National Laboratory and developer for the Globus Project since 1997. His research interests include distributed computing, and distributed and scientific visualization.

UNCORRECTED PROOF

ARTICLE IN PRESS

721

R. Kettimuthu is a graduate student in the Department of Computer and Information at the Ohio State University. His research interests include Grid Computing, High-Performance Networking, Scheduling and Resource Management in Parallel and Distributed Systems. He is currently working on developing efficient techniques for Parallel Job Scheduling. As a student intern at Argonne National Laboratory, he has been working on tuning the transport protocols to improve the performance of bulk data transfer in wide area networks. He received his Bachelor's in Computer Science and Engineering in 1999 from Anna University, India. http://www.cis.ohio-state.edu/∼kettimut.

722

H. Newman (Sc. D, MIT 1974) is Professor of Physics at the California Institute of Technology, and a Caltech faculty member since 1982. He co-led the MARK J Collaboration that discovered the gluon, the carrier of the strong force, at the DESY laboratory in Hamburg in 1979. He has had a leading role in the development, operation and management of international networks and collaborative systems serving the High Energy and Nuclear Physics communities since 1982, and served on the Technical Advisory Group for the NSFNet in 1986. He originated the Data Grid Hierarchy concept and the globally distributed Computing Model adopted by the four LHC high-energy physics collaborations in 1998–2000. He is the PI of the LHCNet project, linking the US and CERN in support of the LHC physics program, a PI of the DOE-funded Particle Physics Data Grid Project (PPDG) and a Co-PI of the NSF-funded International Virtual Data Grid Laboratory. He co-founded and chairs the Internet2 High Energy and Nuclear Physics Working Group, is a member of the Internet2 Applications Strategy Council, and he chairs the Standing Committee on Inter-Regional Connectivity of ICFA (the International Committee on Future Accelerators). He is Chairman of the Board and Co-Founder of VRVS Global Corporation (2001), and has led the US part of the CMS Collaboration (440 physicists at 40 US Institutions) as US CMS Collaboration Board Chair since 1998.

723

S. Ravot is a Network Engineer at the California Institute of Technology, Division of Physics, Mathematics and Astronomy. He is currently based at CERN (Geneva), where he is one of the engineers responsible for the operation of the CERN/US-HENP transatlantic network. In the context of high-speed transatlantic networks he has studied the behavior of TCP over high-bandwidth/latency networks and he is involved in the DataTAG project. He holds a degree in Communication Systems from the Swiss Federal Institute of Technology (Lausanne).

T. Rimovsky is the Assistant Director—Network Engineering and Research, at the National Center for Supercomputing Applications. In this position, he is responsible for production, experimental and research networking at NCSA. He serves on the MREN executive committee and is on the technical teams for I-Wire TeraGrid networking. He has been working in high-performance networking since 1995.

731

C. Steenberg is a Software Engineer in the High Energy Physics Department of the California Institute of Technology, where his research focuses on harnessing the power of Grid and peer-to-peer technologies to empower the next generation of data analysis and modeling tools. He completed a MSc (1995) and PhD (1998) in Physics at the Potchefstroom University in South Africa, on the topic of cosmic ray propagation in the solar system, and continued his research at the Caltech Space Radiation Lab as part of the Voyager program until 2000.

732

L. Winkler is Senior Network Engineer at Argonne National Laboratory's Mathematics and Computer Science Division. She also serves as MREN Technical Director and is a member of the STAR-TAP/StarLight engineering team. Her focus since 1995 has been in the area of interconnectivity and interoperability of wide-area research networks in support of advanced scientific and engineering applications. She earned a BS in Computer Science from Purdue University in 1980, and a MS in Management from Purdue University in 1983.

UNCORRECTED PROOF